Model Validation and Clustering

Gerardo De la O

2022

Model Validation (k-fold XVal and Train/Test splits)

This section explores the evaluation of models using k-fold cross-validation and the more traditional "training-validation-test partition" methods. To provide a thorough demonstration of the aforementioned techniques, we will explore cross-validation with both KNN and SVM algorithms individually. We will then follow with an example of the partition method using both KNN and SVM simultaneously (3 Parts total).

Preparation

```
library(kernlab)
library(kknn)
library(caret)
library(dplyr)

filename = 'credit_card_data.txt'
data <- read.table(paste0('data 2.2/',filename))
```

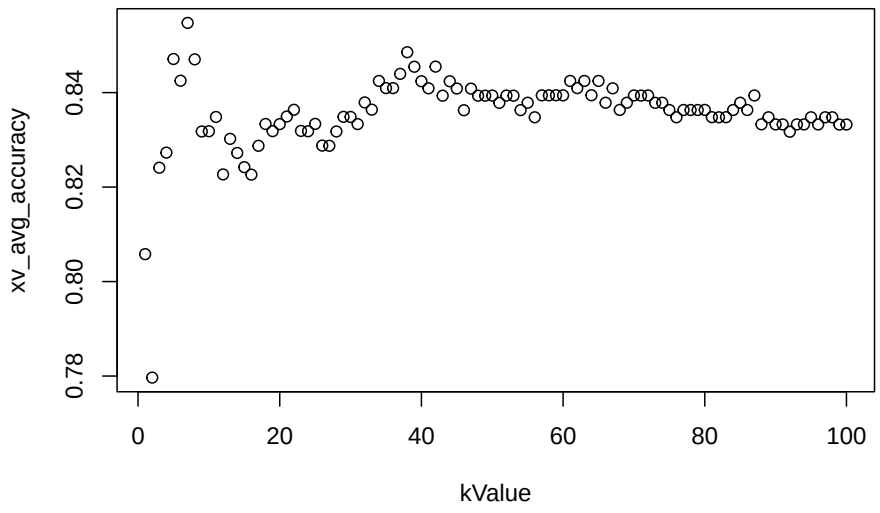
Cross-Validation

K-fold cross-validation is the practice of splitting the data set into K groups, and then iteratively train a model against each of the groups with the goal of obtaining an "Average Performance" metric. Final model parameters will finally be chosen based on the best average.

Part 1 - KNN with 10-fold Cross-Validation This exercise computationally simplifies the knn algorithm from last homework. Since we now have access to well-defined training and testing partitions for each of our k groups, we build the model based on those groups (instead of training each row against the rest of the data set). In the code below, we iterate through a series of k-neighbor values and compute the means of 10-fold cross-validation for each k. The code then automatically chooses the best k to use for the final model.

```
set.seed(0)
#Accumulators of k and evaluation performances
kValue = c()
xv_avg_accuracy = c()

#Cross-Validation Groups: Returns the set of values that are NOT in the group
cv_split <- createFolds(data$V11, k=10, list=TRUE, returnTrain = FALSE)

#Test many #Knn values
for(k in seq(1,100)){
    #Accumulator of each group's performance
    accuracies = c()

    #K-fold XValidation
```

```
        for(i in cv_split){
            #Define train and test sets for each group
            train = data[-i,]
            test = data[i,]

            output = data[i,ncol(data), drop=FALSE]

            model = kknn(V11~., train=train,test=test, k=k, kernel="rectangular", scale='TRUE')
            fitted_vals = model$fitted.values

            evaluation <- output %>%
                mutate(model_pred = 1*(fitted_vals > .50), accurate= 1*(model_pred == V11))
            accuracy <- sum(evaluation$accurate/nrow(evaluation))
            accuracies <- c(accuracies, accuracy)

        }

        performance <- mean(accuracies)
        kValue <- c(kValue, k)
        xv_avg_accuracy <- c(xv_avg_accuracy, performance)

    }

    kMatrix <- data.frame(kValue, xv_avg_accuracy, stringsAsFactors=FALSE)
    plot(kMatrix)
```




```
    # Pick K with the best performance
    bestPerformance <- kMatrix[which.max(kMatrix$xv_avg_accuracy),]
    print(bestPerformance)
```

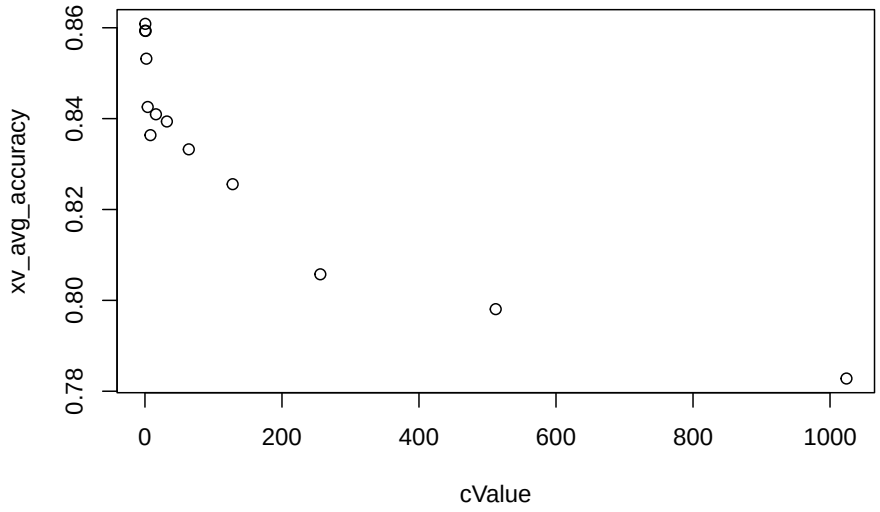
```
##   kValue xv_avg_accuracy
## 7      7       0.8547552
```

```
#finalK = bestPerformance['kValue']
#f_model = kknn(V11~., train=data,test=test, k=finalK, kernel="rectangular", scale='TRUE')
```

Part 2 - SVM with 10-fold Cross-Validation Using the last homework as reference, we found that svm with non-linear kernel was, by far, the most accurate model. We discussed how, due to the kernel trick, a gaussian svm is able to generate non-linear decision boundaries. While this allows the model to fit the data particularly well, it runs the risk of over-fitting. Now, we have a tool to prevent over-fitting! The code below iterates through a list of C values and then finds an optimal value of C using k-fold cross-validation. The final C is then piped in to a final model for subsequent use.

```
set.seed(0)

x <- as.matrix(data[,-ncol(data)])
y <- as.factor(data[,ncol(data)])

#Accumulators of c and the evaluation performance (average of the k subsets)
cValue = c()
xv_avg_accuracy = c()

#Cross-Validation Groups: Returns the set of values that are NOT in the group
cv_split <- createFolds(y, k=10, list=TRUE, returnTrain = FALSE)

#Test C values in powers of 2 increments (.5,1,2,4,8,...)
for( j in c(2^(-2:10))){
    #Accumulator of each group in k
    accuracies = c()

    #K-fold XV
    for(i in cv_split){
        #Define train and test sets for each group
        x_train = x[-i,]
        x_test = x[i,]
        y_train = y[-i]
        y_test = y[i]

        output = data[i,ncol(data), drop=FALSE]

        model = ksvm(x_train, y_train, type = 'C-svc', kernel='rbfdot', C=j,
                     kpar = 'automatic' , scaled=TRUE, class.weights=NULL, prob.model=FALSE)

        #Obtain performance of each group
        predictions = predict(model, x_test, type = "response")

        evaluation <- output %>%
            mutate(model_pred = predictions, accurate= 1*(model_pred == V11))
        accuracy = sum(evaluation$accurate/nrow(output))
        accuracies <- c(accuracies, accuracy)
    }

    #Calculate overall performance for each C, store in output table
    performance = mean(accuracies)
    cValue <- c(cValue, j)
    xv_avg_accuracy = c(xv_avg_accuracy, performance)
```

```
    }

    cMatrix = data.frame(cValue, xv_avg_accuracy, stringsAsFactors=FALSE)
    plot(cMatrix)
```




```
    # Pick C with the best performance
    bestPerformance = cMatrix[which.max(cMatrix$xv_avg_accuracy),]
    finalC = bestPerformance['cValue']
    print(bestPerformance)
```

```
##   cValue xv_avg_accuracy
## 2    0.5       0.8608625
```

```
    #Run final model against full data set
    f_model = ksvm(x, y, type = 'C-svc', kernel='rbfdot', C=finalC,
                   kpar = list(), scaled=TRUE, class.weights=NULL, prob.model=FALSE)
    x = xmatrix(f_model)[[1]]
    c = coef(f_model)[[1]]

    #Ax + a0 = 0 (our linear equation)
    a <- colSums(x * c)
    a0 <- b(f_model)

    a['a0'] <- a0
    a
```

```
##          V1          V2          V3          V4          V5          V6
##   4.7175804  37.4332111  36.5028160  74.8879510 138.6648492 -72.9327967
##          V7          V8          V9         V10          a0
##  88.1332197  -4.5630860 -22.7408550  46.7282087  -0.3616408
```

Results

Based on the work above, the best models based on 10-fold cross-validation are as follows:

```
1. For KNN:
    1. K = 7
    2. Cross validation average performance: 85.4%
2. For SVM:
    1. Kernel: Gaussian ('rbfdot')
    2. C = 0.5
    3. Cross validation average performance: 86.1%
```

Traditional Train, CV, Test Split

This part explores a more traditional data splitting process. We will split the initial data set once, and repeatedly test different parameters against the validation set to evaluate performance. Note that this approach is similar to running k-fold validation with k=1! On the other end of the spectrum, k-fold validation will approach "Leave-one-out validation" (LOOCV - not covered), as k approaches the number of data points.

Part 3 - KNN & SVM with train-validation-test split The following code evaluates a series of K and C values for both KNN and SVM. Then, it will automatically choose the best model and parameter.

```
set.seed(1234)
#Set percentage of 1st and second splits
samplePct1 <- .7
samplePct2 <- .5

#Split the data
trainSplit <- sample(seq_len(nrow(data)), size = floor(samplePct1 * nrow(data)))
train <- data[trainSplit , ]
test <- data[-trainSplit , ]

testSplit <- sample(seq_len(nrow(test)), size = floor(samplePct2 * nrow(test)))
validation <- test[testSplit , ]
test <- test[-testSplit , ]

#Accumulator of model type, k (or C) and accuracy values
kValue = c()
accuracies = c()
modelType = c()

#Test #Knn K values
for(k in seq(1,100,1)){
    output = validation[,ncol(validation), drop=FALSE]

    model = kknn(V11~., train=train,test=validation, k=k, kernel="rectangular", scale='TRUE')
    fitted_vals = model$fitted.values

    evaluation <- output %>%
        mutate(model_pred = 1*(fitted_vals > .50), accurate= 1*(model_pred == V11))
    accuracy = sum(evaluation$accurate/nrow(evaluation))

    accuracies <- c(accuracies, accuracy)
    kValue <- c(kValue, k)
    modelType <- c(modelType, 'knn')

}
```

```
    #test svm C values
    for(c in seq(1,100,1)){
        output = validation[,ncol(validation), drop=FALSE]

        model = ksvm(V11~.,data=train, type = 'C-svc', kernel='rbfdot', C=c,
                     kpar='automatic', scaled=TRUE, class.weights=NULL, prob.model=FALSE)

        predictions = predict(model, validation[,1:10], type = "response")
        evaluation <- output %>%
            mutate(model_pred = predictions, accurate= 1*(model_pred == V11))

        accuracy = sum(evaluation$accurate/nrow(output))
        accuracies <- c(accuracies, accuracy)
        kValue <- c(kValue, c)
        modelType <- c(modelType, 'svm')

    }

    kMatrix <- data.frame(modelType, kValue, accuracies, stringsAsFactors=FALSE)

    # Pick Model with the best performance
    bestPerformance <- kMatrix[which.max(kMatrix$accuracies),]
    finalK = bestPerformance[1,'kValue']

    #Run final model against test data
    f_output <- test[,ncol(test), drop=FALSE]

    f_model <- kknn(V11~., train=train,test=test, k=finalK, kernel="rectangular", scale='TRUE')
    fitted_vals <- f_model$fitted.values

    evaluation <- f_output %>%
        mutate(pred = 1*(fitted_vals > .50), accurate= 1*(pred == V11))
```

Traditional Split - Results

Based on the work above, the best model based on validation set performance is as follows:

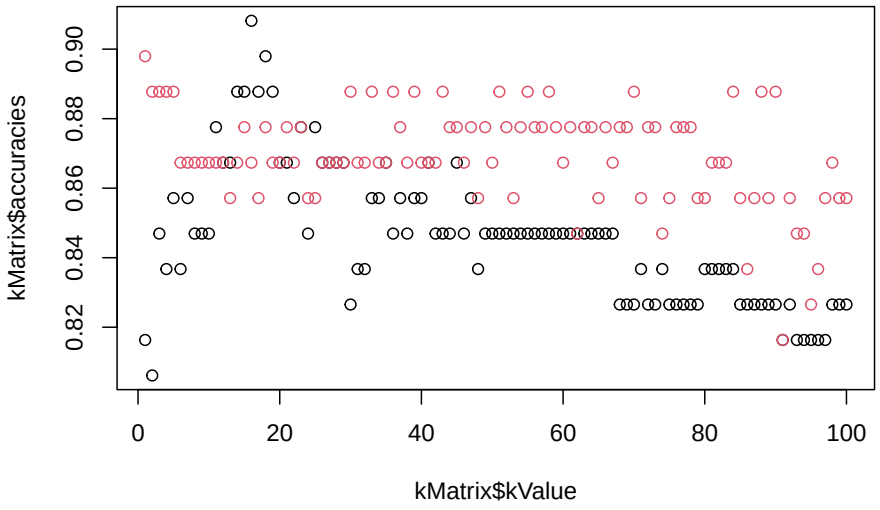
1. Model Type: KNN (black dots in the plot below)
2. Parameter: K = 16
3. Validation performance: 90.8%
4. Test Performance: 83.8%

Note that the Test Set performance is considerably lower than the Validation Set performance. While this method is simpler, the process would benefit by reducing the variance of our individual evaluations (which k-fold cross validation does).

```
print(bestPerformance)
```

```
##    modelType kValue accuracies
## 16       knn     16  0.9081633
```

```
plot(kMatrix$kValue, kMatrix$accuracies, col=as.factor(kMatrix$modelType))
```




```
confusion <- confusionMatrix(data=as.factor(evaluation$pred), reference = as.factor(evaluation$V11))
confusion
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 49  7
##          1  9 34
##
##                Accuracy : 0.8384
##                  95% CI : (0.7509, 0.9047)
##     No Information Rate : 0.5859
##     P-Value [Acc > NIR] : 6.121e-08
##
##                   Kappa : 0.6693
##
##  Mcnemar's Test P-Value : 0.8026
##
##             Sensitivity : 0.8448
##             Specificity : 0.8293
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.7907
##              Prevalence : 0.5859
##          Detection Rate : 0.4949
##    Detection Prevalence : 0.5657
##       Balanced Accuracy : 0.8370
##
##        'Positive' Class : 0
##
```

Real World Example - Unsupervised Learning

I currently work at a B2B technology company specializing in conversational analytics. From a a business perspective, clustering can help our client companies gain insight into common themes that occur in their phone conversations and chat interactions. A very impactful use case of clustering models, for example, would be to group "reasons for contact" in order to determine appropriate diagnosis and resolution paths. For this exercise, we will assume that we have an ideal set of predictors available to us, and will be attempting to capture groups that include metadata outside of the interaction transcript itself.

Predictors that I would consider are:

1. Demographic information about the caller (age, location, income, gender, etc.)
2. Consumer metadata (account type, customer tenure, products purchased, LTV, etc.)
3. IVR responses (IVR is the automated menu provided prior to the interaction)
4. Disposition (Disposition codes are agent labels and comments added after the interaction)
5. Phrases used at the beginning of the interaction

After modeling and analysis, examples of potential findings would be something like "people over age 50 tend to call in about technical issues with the product they purchased" or "people who signed up during a promotion period are likely to inquire about cost". Based on those types of insights, the company would be able to confidently implement policy to optimize their issue resolution.

Clustering

This sections explores the use of k-means clustering to group species of flowers. Optimal input variables and k value are briefly discussed, and performance of the clustering algorithm is evaluated against actual responses.

Preparation

```
library(datasets)
data('iris')

head(iris,2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

K-Means

In this section, we use the K-means clustering algorithm to best group the "iris" data det in base R. Note that since this is an unsupervised model, we don't need a test set. Instead, we will directly use the output "Species" to directly evaluate results at different k values.

```
set.seed(5)
#Accumulator of model type, k (or C) and accuracy values
kValue = c()
accuracies = c()

x <- as.matrix(iris[,-ncol(iris)])
#x2 <- select(iris, -Species, -Sepal.Width)
y <- as.factor(iris[,ncol(iris)])

#Scaling performs Worse in this case. I chose to omit
#x <- scale(x,center=TRUE, scale=TRUE)
for(k in seq(1,7)){
```

```
        model = kmeans(x, centers=k)

        cluster_membership = model$cluster
        tot_distance = model$totss
        tot_within_dist = model$tot.withinss
        cluster_size = model$size

        kValue <- c(kValue, k)
        accuracies <- c(accuracies, tot_within_dist)

        cm <- table(iris$Species, cluster_membership)
        print(cm)

    }
```

```
##             cluster_membership
##               1
##   setosa     50
##   versicolor 50
##   virginica  50
##             cluster_membership
##               1  2
##   setosa     50  0
##   versicolor  3 47
##   virginica   0 50
##             cluster_membership
##               1  2  3
##   setosa     50  0  0
##   versicolor  0 48  2
##   virginica   0 14 36
##             cluster_membership
##               1  2  3  4
##   setosa      0 50  0  0
##   versicolor  0  0 27 23
##   virginica  27  0  1 22
##             cluster_membership
##               1  2  3  4  5
##   setosa      0 17  0 33  0
##   versicolor 23  0  0  0 27
##   virginica  17  0 32  0  1
##             cluster_membership
##               1  2  3  4  5  6
##   setosa      0  0  0  0  0 50
##   versicolor  7  2 21  0 20  0
##   virginica   0 27  0 22  1  0
##             cluster_membership
##               1  2  3  4  5  6  7
##   setosa      0  0  0 22  0 28  0
##   versicolor 27  4 19  0  0  0  0
##   virginica   1 15  0  0 22  0 12
```

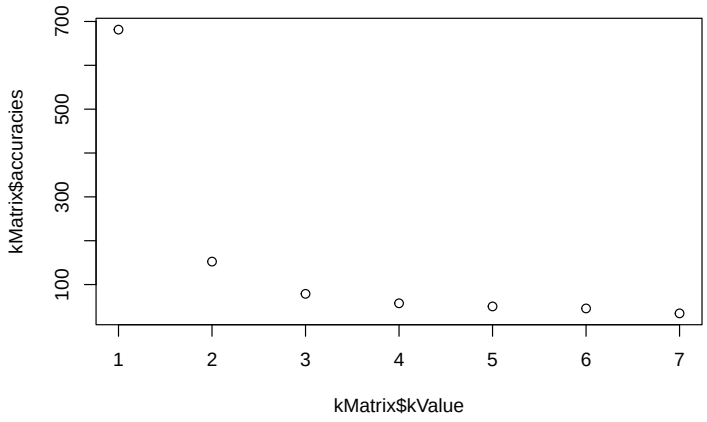
```
    kMatrix = data.frame(kValue, accuracies, stringsAsFactors=FALSE)
```

Clustering Results

Based on the work above:

1. Best combination of predictors:
    a. Either all of them, or exclude 'Sepal.Width' (no conclusive evidence).
    b. The residual decreased when Sepal.Width was removed, but the predictions didn't change.
    c. See box plot below to see how Sepal.Width is mostly overlapping for 2 of the classes.
2. Suggested value of k:
    a. 3
    b. This is fairly obvious given that we have the true outcomes.
    c. Assuming we didn't know this, however, the Elbow Method shows a clear optimum at 3
3. How well does the best model predict flower type:
    a. Setosa was predicted perfectly at 100% accuracy
    b. Versicolor was predicted very well at 96% accuracy. 2 samples misclassified as Virginica
    c. Virginica was predicted at 72% accuracy. 14 samples misclassified as Versicolor

```
kMatrix = data.frame(kValue, accuracies, stringsAsFactors=FALSE)
plot(kMatrix$kValue, kMatrix$accuracies)
```




```
boxplot(Sepal.Width~Species, data = iris)
```